TRAFFIC GENERATION AND UNIX/LINUX SYSTEM TRAFFIC CAPACITY ANALYSIS

James F Brady
Capacity Planner for the State Of Nevada
jfbrady@doit.nv.gov

Often requirements dictate that a transaction based application be capacity tested before it is released into production using a traffic generator that simulates customer requests against a target system. Many times the feedback obtained from the traffic generator is insufficient to adequately quantify the quality of the offered load produced and the analysis performed on the target system resource consumption levels is insufficient to clearly illustrate traffic capacity. This paper provides some suggestions for better quantifying traffic generation quality and more clearly illustrating traffic capacity when the target system is running in a Unix/Linux environment.

1. Introduction

Often requirements dictate that a transaction based application be capacity tested before it is released into production using some type of traffic generation mechanism that simulates customer requests to a target system. Many times the traffic generation portion of such tests yields insufficient information to quantify the offered load's conformance to real world traffic conditions and the statistical analysis of resource utilization levels performed on the target system does not clearly illustrate traffic capacity. This paper is intended to provide some suggestions for better quantifying traffic generation quality and more clearly illustrating traffic capacity when the target system is running in a Unix/Linux environment.

Whether one is traffic capacity testing a transaction oriented application as part of the product development process or demonstrating compliance with a capacity requirement the usual approach is to acquire a shareware or commercial traffic generation tool. This tool is adapted to the situation at hand and run to produce a stock set of graphs which show resource consumption levels as a function of virtual users. Figure 1 is an example of one such set of graphs for mean response time and CPU utilization.

Virtual users are normally modeled as session scripts like those represented in the Figure 2 matrix. The virtual user session mix scenario shown is Web based with transactions consisting of a login followed by queries and/or updates and a logoff. A think time delay is imposed between each script step in the five separate virtual user scripts shown.

Figure 1: RT and CPU Vs Virtual Users

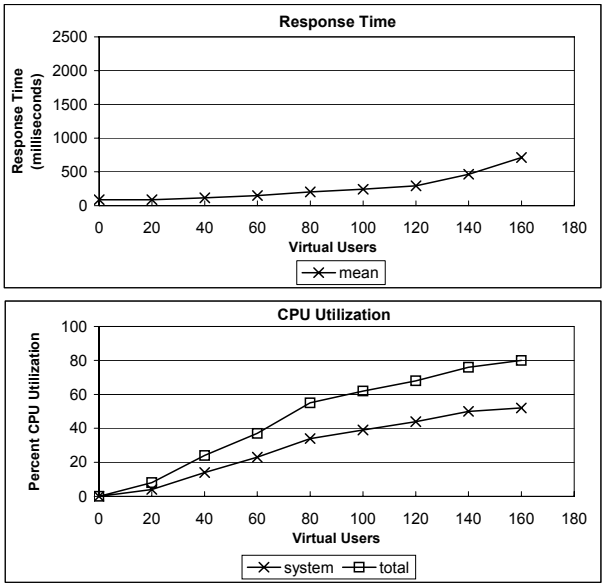


Figure 2: Virtual User Mix Scenario

Transaction	Delay Seconds	Vuser Type 1	Vuser Type 2	Vuser Type 3	Vuser Type 4	Vuser Type 5	Total
LOGIN	10	1	1	1	1	1	100%
Query1	15	1		1		1	75%
Query2	10		1		1	1	50%
Update1	20	1					10%
Update2	15		1				15%
LOGOUT	10	1	1	1	1	1	100%
	ScriptTime	55	45	35	30	45	40.5

Usually, traffic is ramped up by adding sets of virtual user scripts until it is determined that a particular resource is exhausted or a service level objective is exceeded. The number of active virtual users at this saturation point is declared to be the capacity of the target system.

This virtual user approach to traffic capacity specification and testing provides a basic understanding of the target system's resource limitations. There are a couple of questions which, if answered, will greatly enhance the quality of the test performed and add to its credibility.

- Does the traffic generator being used present a real world traffic pattern to the target system and, if so, how can that pattern be quantified?
- Because the target system processes transactions, not virtual users, what transaction rate and mix does each virtual user level tested represent?

The answer to the first question significantly enhances test quality and the response to the second question permits better identification of target system bottlenecks and provides data to more clearly show that system's traffic capacity.

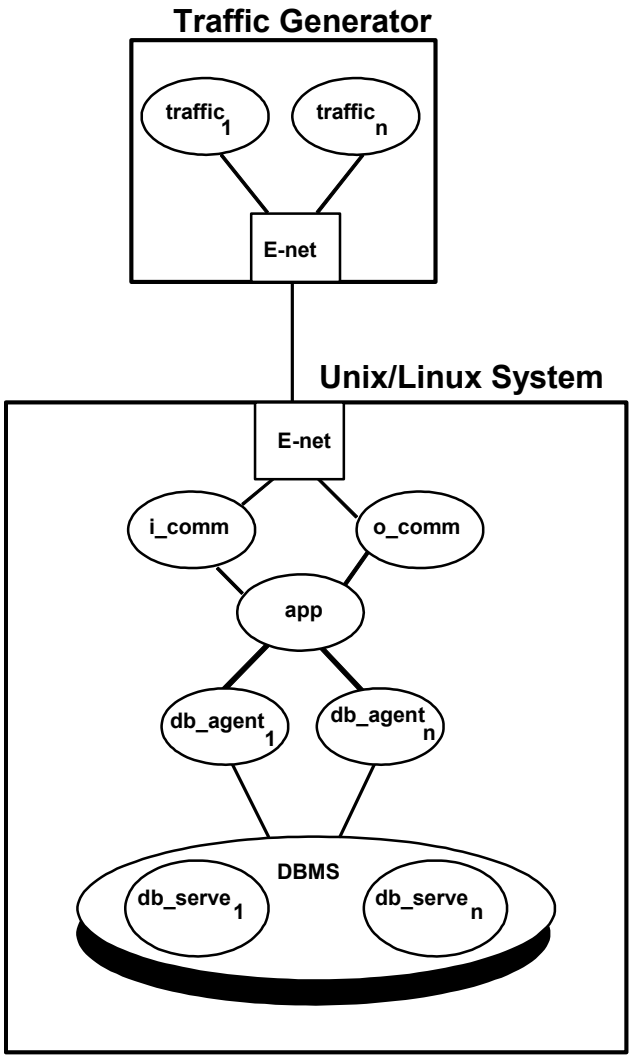
The approach taken in this paper to address these two questions is, first, describe an example traffic capacity testing environment and, second, use that environment as a foundation for traffic generation quality and target system traffic capacity discussions. These two discussions then lead to a list of observations and an associated set of recommendations. Some concluding remarks and summary comments along with a traffic generation methodology appendix complete the paper.

2. Traffic Capacity Test Environment

Figure 3 shows the layout of the example Traffic Generator and associated target Unix/Linux System from a process flow perspective. The traffic generator in this example is also Unix/Linux based and has "n" processes producing traffic for the target environment over an Ethernet connection.

The flow of transactions in this situation occurs in the following way. The target system's *i_comm* process receives transaction data initiated by one of the traffic generator's "n" traffic processes over the Ethernet interface, reformats it and places it on the *app* process's message queue. The *app* process executes logic and constructs a message that it deposits on one of the "n" *db_agent* process message queues. The selected *db_agent* formulates an SQL request and submits it to the DBMS environment. One of the DBMS *db_serve* processes retrieves the required data and makes the data available to the requesting *db_agent* which passes it forward to the *app* process. The *app* process builds a response and hands it to the *o_comm* process where it is reformatted and sent over Ethernet to the traffic process initiating the request. The traffic process analyzes the response to determine if it is correct and records the transaction response time.

Figure 3: Traffic Generation Topology



This example transaction traffic flow is primarily intended for illustrative purposes but can be viewed as the typical Web Server, Application Server, and Data Base Server implementation in a single serving system. The *comm* processes perform the Web Server tasks, the *app* process handles the Application Server logic, and the *db_agen*t along with the DBMS environment provide the Database Server functionality. Here message queues are used to communicate between functional components instead of the communications links associated with the three server arrangement.

With this overall environment in mind the next section

focuses on the portion of Figure 3 that simulates customer requests against the target system, the Traffic Generator. The emphasis in this discussion is traffic generation quality and identification of needed feedback mechanisms to insure that quality is maintained throughout the testing period.

3. Traffic Generation

Traffic generation is usually accomplished with a computer program running on one or more simulation computers that launch requests, measure the time for responses to occur, and determine if responses are correct. Traditionally, virtual user script based traffic generators, exemplified by Figure1 and Figure 2, are implemented for this purpose but the one used in this example, developed by the author, is set up differently.

The traffic generator implemented in this illustration is transaction oriented, not virtual script based, and has the advantage that the request statistics produced are in the units of work presented to the target system, i.e., transactions/sec. This transaction model can be used for traffic generation when connectionless protocols like Web HTTP are being exercised. Further analysis is necessary when using this technique to map traffic volume to users supported.

Figure 4 is an output screen from this traffic generator which reflects a run lasting 12 minutes where good, bad, and late request statistics are reported every 100 seconds for both GET and POST Web queries. The traffic mix implemented for this run is shown in Figure 5 as percentages of queries and updates.

The software is designed such that each process or thread always makes the same type of transaction request. In this example, each requesting process performs its assigned query or update after it has first successfully completed a single login.

The traffic run in Figure 4 includes 100 processes creating traffic with the Figure 5 mix. The 50 GET and 50 POST processes yield an aggregate transaction rate of slightly less than 10 transactions per second with a mean delay between requests of approximately 10 seconds, i.e., 9996 milliseconds.

The traffic generator in Figure 4 produces its traffic consistent with the most common real world environment used for simulating transaction processing traffic which is request independence, or random arrivals. The random arrivals environment is referred to in the literature as a Poisson process and is quantified by two formulas from probability theory. The times between arrivals (t) are negative exponentially distributed and the number of arrivals in constant length intervals are Poisson distributed. Therefore, the 9996 milliseconds is the mean of a negative exponentially distribution set of delay times.

For a detailed discussion of how traffic is generated with these statistical properties see reference [BRAD04] and Appendix A at the end of this paper.

Figure 4: Traffic Generator Run

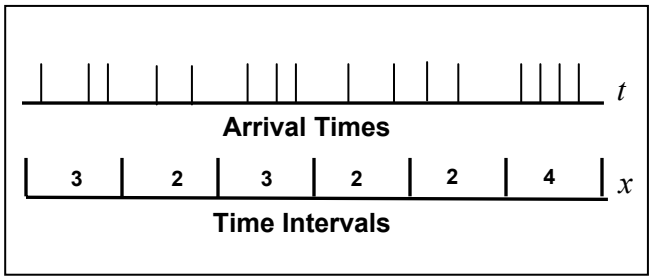
```
Linux Sat Feb 4 12:05:43 2006, 50 GET and 50 POST source(s), 3000 ms late
   ./web_traffic -t -i 100 -s 720 9996 perf_measure_100

                                  --------GET-------      -------POST-------
  Time       sent     !recv      good     bad   late      good     bad   late
12:05:43      949         0       473       0      0       476       0      0
12:07:23      970         0       453       0      0       517       0      0
12:09:03     1000         0       503       0      0       497       0      0
12:10:43      966         0       475       0      0       491       0      0
12:12:23     1010         0       494       0      0       516       0      0
12:14:03      937         0       443       0      0       494       0      0
12:15:43     1005         0       482       0      0       523       0      0
   Total     6837         0      3323       0      0      3514       0      0


Caught a SIGALRM signal -- shutting down


Sat Feb 4 12:16:18 2006
```

Figure 5: Transaction Based Traffic Mix

Transaction	Percent
LOGIN	Once Per Source
Query1	50%
Query2	33%
Update1	7%
Update2	10%
LOGOUT	0%
Total	100%

Figure 6, is a pictorial representation of a random arrivals pattern where the arrival times, t, are represented by the non-uniform vertical bars and the frequency counts, x, are the numbers indicating the arrivals within the uniform intervals shown. Mathematically, (t) is described by the negative exponential probability density function, Equation 3.1, which has a mean inter-arrival time of μ. The number of arrivals (x) in constant length intervals is mathematically illustrated as the Poisson probability density function, shown as Equation 3.2, which has a mean number of arrivals per interval of λ.

Figure 6: Random Arrivals



$$f(t) = \frac{1}{\mu} e^{-\frac{t}{\mu}} \qquad (3.1)$$

$$f(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \qquad (3.2)$$

Where:

$\mu = mean\ inter\text{–}arrival\ time\ t$.

$for\ f(t),\quad \mu = mean = std\ dev = \sqrt{variance}$.

$\lambda = mean\ number\ of\ arrivals\ per\ interval\ x$.

$for\ f(x),\quad \lambda = mean = variance$.

Estimates of mean $\bar{x}$, variance s^2, and standard deviation s are:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n},\quad s^2 = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n-1},\ s = \sqrt{variance}.$$

It is interesting to note that the mean of the negative exponential distribution is equal to its standard deviation and the mean of the Poisson distribution is equal to its variance. Formulas for estimating mean, standard deviation, and variance are provided above.

These parameter estimation formulas can be used to analyze arrival data and make judgments concerning the likelihood the arrivals are random. The arrival summary statistics report in Figure 7 show these characteristics for each of the eight traffic runs listed. The traffic run pass 0200 of Figure 7 maps to the Figure 4 run and shows a mean and standard deviation inter-arrival time of 101.37 milliseconds and 99.05 milliseconds, respectively.

Figure 7: Arrival Summary Statistics

Arrival Summary Statistics (ms) - Demo System Saturday 02/04/2006

pass	n	tps	median	mean	sdev	min	max
1000	1321	2.20	304	452.80	456.81	0	3573
0300	3936	6.56	103	151.42	157.39	0	1468
0200	5867	9.78	72	**101.37**	**99.05**	0	1011
0130	8866	14.78	47	66.82	67.33	0	717
0110	10223	17.04	40	57.88	58.01	0	556
0100	11147	18.58	37	53.01	52.53	0	448
0090	12613	21.02	32	46.79	47.37	0	502
0085	13472	22.45	30	43.76	43.94	0	388

Therefore, the best feedback for quantifying the quality of the offered load produced by a traffic generator creating random arrivals traffic is to compute inter-arrival time mean and standard deviation. If these two statistics are close to the same value, traffic generation quality is good but if they are not, traffic generation quality is suspect and can yield misleading traffic capacity results.

Presumably, traffic generators which use the techniques exemplified in Figure 2 provide transaction arrival pattern statistics like those contained in Figure 7. The virtual user script mechanism represented by Figure 2 is unlikely to produce a random arrivals traffic pattern, however, because the think times are fixed values, or if "random delay" is an option, it is usually specified as a minimum and maximum range of delay times. This specification is not consistent with the negative exponential distribution which contains one parameter, the arithmetic mean.

Whether a traffic generator is virtual script or transaction based, statistical feedback regarding the quality of its request launch pattern is important. The test results obtained from a traffic generator that does not provide inter-arrival time statistics or whose inter-arrival mean and standard deviation are available but not close to each should be scrutinized very carefully.

It also seems that the virtual user script environment is one where a consistent and representative transaction traffic mix may be difficult to maintain throughout the example eight runs represented in Figure 7. This is likely because it is hard to anticipate transaction mix from sets of running script combinations. Transaction mix consistency is important for bottleneck identification, which is a topic discussed in the next section.

The eight traffic runs reflected in the Figure 7 arrival statistics yield the target system response time statistics shown in Figure 8. Note the mean and standard deviation of response times are close in value and therefore appear to be negative exponential in nature. It is the author's experience that this phenomenon occurs quite often even though there is no causal relationship between response times and the negative exponential as there is for inter-arrival times.

Figure 8: Response Time Summary Stats

Response Time Summary Stats (ms) - Demo System Saturday 02/04/2006

pass	n	tps	mean	sdev	p95	min	max
1000	1322	2.20	87.92	82.27	253	1	613
0300	3937	6.56	116.45	111.97	343	1	903
0200	5868	9.78	**150.22**	**142.62**	434	1	1423
0130	8867	14.78	203.23	198.99	603	1	1623
0110	10224	17.04	242.02	234.59	723	1	1983
0100	11148	18.58	291.49	293.35	883	1	2823
0090	12614	21.02	463.68	457.80	1383	1	4742
0085	13473	22.45	712.08	710.31	2143	1	7884

Now that traffic generation quality indicators have been identified the next section focuses on traffic capacity analysis of the Unix/Linux Target System portion of the Traffic Generation Environment in Figure 3. This analysis uses the eight offered traffic increments contained in the Figure 7 Arrival Summary Statistics Table and in the corresponding Figure 8 Response Time Summary Stats Table.

4. Target System Traffic Capacity

The most important outcome of the target system resource consumption analysis is to clearly illustrate traffic capacity and identify any system imbalances that exist. Often the best way to show a system's capacity characteristics is to draw a picture of resource consumption levels and response time service levels

as a function of incremental increases in traffic volume. The graphs in Figure 9 are such a picture and show response time and resource consumption levels for CPU, Disk, and Network Packets as a function of transactions per second.

These graphs are the end result of a systematic traffic capacity testing procedure where traffic is generated as described in Section 3 and resource consumption information is collected and analyzed on the target Unix/Linux System using tools developed by the author. These tools invoke and process data gathered by standard Unix/Linux metering utilities; i.e., sar, vmstat, iostat, netstat, ps, and Linux /proc system counters. Each of the eight traffic rate levels shown are produced by sampling resource consumption levels every 15 seconds over a twelve minute period with the negative effects of startup and shutdown mitigated by restricting the analysis to the middle ten minutes.

Response time in Figure 9 is expressed in terms of mean and 95% level with CPU Utilization divided into operating system and total percentages. The 95% level of response time provides an important indication of response time variability not available from the mean value and greatly enhances understanding of the target system's service level characteristics.

It is interesting to contrast the CPU Utilization graph in Figure 9 with Figure 1. Figure 9 is a graph of CPU utilization versus transactions per second and Figure 1 is a plot of CPU utilization against virtual users. If traffic mix is held constant for both scenarios across traffic volume runs, a balanced system yields a straight line relationship for Figure 9 but does not for Figure 1. The Figure 1 curve is non-linear because an incremental increase in virtual users adds less then a proportional amount of offered traffic to the target system since virtual user transaction response times are elongated. Therefore, whenever possible, it is useful to plot resource utilization levels as a function of traffic rate to determine if bottlenecks exist in resources such as CPU, Disk, and Network interfaces. The linear relationship between these three resources and the transaction rate for all test runs indicates they are balanced resources and not bottlenecks.

The pie chart in Figure 10 provides an indication of CPU Utilization from a running process perspective as opposed to the resource consumption orientation of Figure 9. This graph shows CPU utilization levels for the processes outlined in the Figure 3 Traffic Generation Topology. It is created by proportioning the CPU time consumed by these processes during the four highest traffic test runs performed. The Figure 11 table that follows the pie chart is the process analysis output from these four traffic runs.

Figure 9: Target System Traffic Capacity

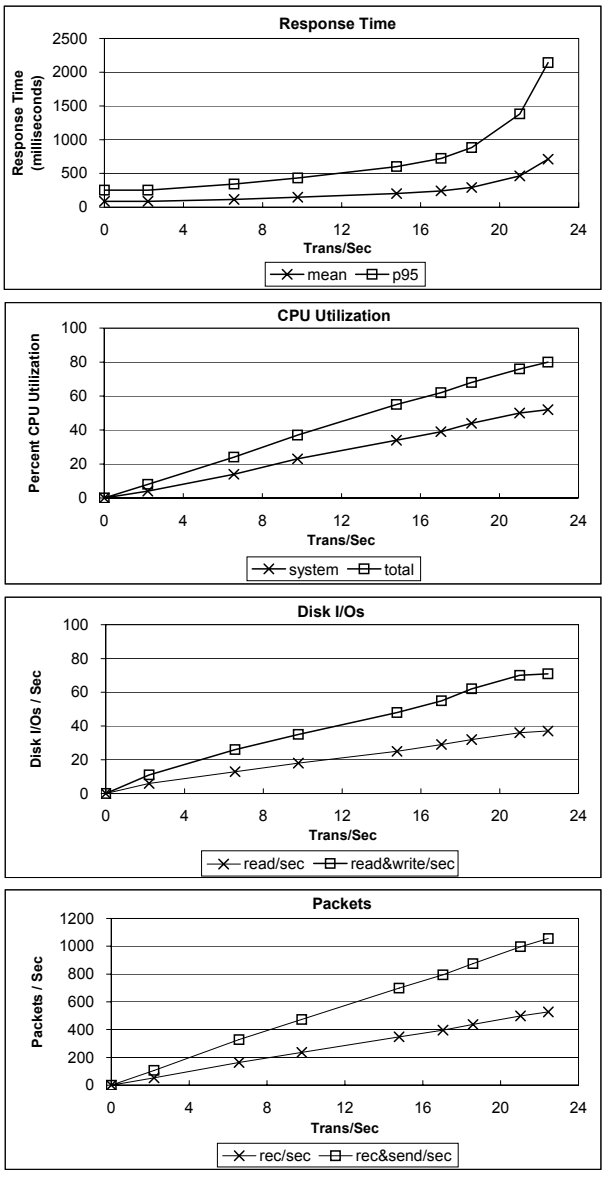


Figure 10: Process Pie Chart

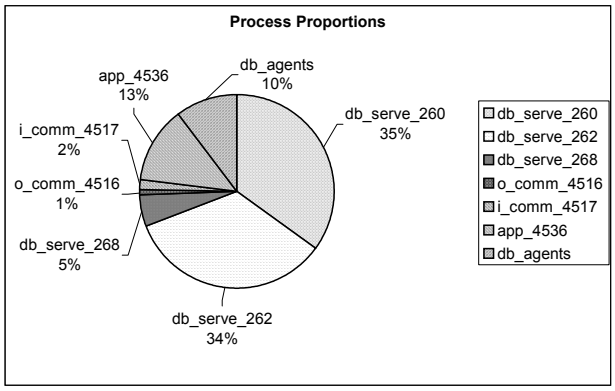


Figure 11: Process Proportions Table

Process Status Summary Statistics - Demo System Saturday 02/04/2006

		--0110---		--0100---		--0090---		--0085---		----Sum-----	
name	pid	sec	%	sec	%	sec	%	sec	%	sec	%
db_serve	260	319	35	357	35	406	35	424	35	1506	35.04
db_serve	262	316	34	347	34	391	34	408	34	1462	34.02
db_serve	268	49	5	53	5	60	5	64	5	226	5.26
o_comm	4516	11	1	11	1	12	1	13	1	47	1.09
i_comm	4517	14	2	15	1	17	1	19	2	65	1.51
app	4536	116	13	128	13	149	13	163	13	556	12.94
db_agent	4542	14	2	17	2	20	2	21	2	72	1.68
db_agent	4548	15	2	17	2	19	2	20	2	71	1.65
db_agent	4549	16	2	18	2	19	2	20	2	73	1.70
db_agent	4550	16	2	18	2	20	2	20	2	74	1.72
db_agent	4551	16	2	17	2	20	2	20	2	73	1.70
db_agent	4554	16	2	17	2	19	2	21	2	73	1.70
	total	918	102	1015	101	1152	101	1213	102	4298	100.00

In the previous section it was suggested that traffic generation quality can be judged by determining how close to equal the mean and standard deviation of the transaction inter-arrival times are to each other. In this section it is shown that graphing resource consumption levels against traffic rates makes traffic capacity levels clearer and helps identify bottlenecks. The next section contains some further observations regarding traffic generation quality and target system capacity determination.

5. Observations

The following are observations which result from and expand on the discussion thus far.

1. Traffic pattern measurements like those in Figure 7 are an important quality indicator for any traffic capacity test and transaction rates are needed to produce clear and concise traffic capacity analysis graphs such as those in Figure 9.

2. Traffic generating software is typically implemented on a Windows XP or Unix/Linux computer which simulates user request timing by having the assigned thread or process call a sleep function. The Unix/Linux traffic generator represented in Figure 4, for example, calls nanosleep() for each negative exponentially drawn GET or POST request inter-arrival time. The operating system awakens the affected thread or process when the sleep time expires and puts it on the ready to run list for execution at its dynamically assigned priority. If the traffic generator is relatively idle, execution occurs close to the requested time but if the traffic generator is heavily loaded execution may be delayed for a significant period of time skewing the traffic pattern from the original intent.

3. It seems that many of the virtual user script based traffic generators described in Section 1 are set up (at least by default) with no exponential inter-arrival mechanism for starting script sequences or simulating think time delays between script transaction requests. Typically, a significant number of scripts started under these circumstances will cause the traffic generator to produce transaction requests in batches rather than in the desired asynchronous manner.

4. If a particular target environment requires multiple traffic generators for adequate loading, a set of generators which individually produce negative exponential inter-arrival times, Figure 7, will yield a single stream of traffic with the same arrival pattern when their traffic is merged. This “memoryless” property of the negative exponential permits traffic to be increased by adding traffic generators while maintaining traffic pattern integrity.

5. Poor quality traffic pattern has its greatest detrimental impact on user response time estimation and can mislead application developers into erroneously thinking they need to change the sizes of resources like buffers and inter-process communication message queues.

6. The virtual user traffic mix in Figure 2 is very common but seems a bit unrealistic because a user login and logout are performed for every script execution. Users generally login a few times a day and seldom logout. They usually just quit the Web browser.

7. Both the virtual user script traffic generator of Figure 1 and the traffic generator represented in Figure 4 operate in a closed queuing system environment. That is, each thread or process created is a single traffic source that makes a transaction request, waits for the response, and delays for some “think” time before making the next request. Since response time is a portion of the overall delay time between requests its probability distribution can influence the request traffic pattern. If response times are negative exponentially distributed, as in Figure 8, they are memoryless. If they are not negative exponentially distributed but small relative to the “think” time, their impact on the request traffic pattern is mitigated.

8. Even though the focus in this paper is on Unix/Linux target systems, the Windows environment provides the support to construct Figure 9 as well. In fact, the author has produced that picture many times for Windows based target systems using the standard “perfmon” counter log template tool which, when executed, creates a “.csv” file of resource consumption statistics.

6. Recommendations

The above observations in conjunction with the comments made in the earlier sections lead to the following list of recommendations.

1. Collect performance statistics on the traffic generator as well as the target system or systems. All traffic capacity studies performed using the Unix/Linux traffic generator in Figure 4 include a set of Figure 9 graphs for the traffic generator as well as those for the target systems. One useful traffic capacity test cross-check is to see if the packet rates between the traffic generator and the target system to which it is sending and receiving are approximately the same but in transmit/receive reverse order.

2. Whether performing the traffic capacity study or receiving one from a vendor the following items should be included in the results report.
   - ✓ A performance summary that lists capacity statistics such as maximum transaction rates, resource consumption levels, and response time service levels
   - ✓ Figure 3: Traffic Generation Topology
   - ✓ Figure 7: Arrival Summary Statistics
   - ✓ Figure 8: Response Time Summary Stats
   - ✓ Figure 9: Traffic Capacity Graphs for the traffic generator as well as all target systems
   - ✓ Figure 10: Process Pie Chart
   - ✓ Hardware and Software Configurations for the traffic generator and all target systems
   - ✓ The transaction traffic mix used throughout the study
   - ✓ An estimate of users supported based on the traffic mix.

   These items make clear the test environment and the results obtained so future tests on enhanced software and faster platforms can be put in proper context.

3. A completed traffic capacity study should consist of a hierarchy of information applicable to both executives and analysts. The results report in Recommendation 2 provides management with a high level view of the system's capacity status. The response time data and resource consumption information contained in the Unix/Linux stat data, e.g., vmstat, or Windows perfmon output should be available for analysts to review and use as a performance improvement feedback mechanism.

4. When looking for traffic generator products, focus on those that possess the following functionality:
   - ✓ Provides real time feedback on transaction event counts and service levels being achieved. The traffic generator in Figure 4 is set up to report GET and POST transaction counts every 100 seconds during each 720 second run and identify the number of response times that exceed three seconds as late. This real time feedback is especially useful when performing failover tests at high traffic volumes.
   - ✓ Possesses a mechanism to be sure the correct data is returned from a transaction query. Many Web applications, for example, default fail to the login page but the Figure 4 traffic generator performs a string comparison to be sure the Web page returned is correct.
   - ✓ Has the capability to randomly select account specific data from a large list of accounts. This functionality provides a more real world environment by defeating the I/O subsystem cache a significant percent of the time thus allowing disk activity to be observed and measured. One resent test using the Figure 4 traffic generator involved random selection from 10,000 account ID values which caused the needed disk activity to occur.

5. Perform traffic capacity studies by running at each traffic rate for a significant period of time like the example twelve minutes and systematically increase the traffic rate in this manner until resource limits are reached. During the analysis portion of the study mitigate the negative effects of startup and shutdown by restricting the data for each traffic rate to the centered time period which is ten minutes in the example provided.

6. Whenever available, report service level percentile data along with mean value information. The mean response time is a very useful statistic but the 95% response time helps make response time variability more explicit.

7. Conclusions

Most of the products available that are designed to create traffic for traffic capacity studies are structured around the idea of virtual user scripts. The emphasis of these tools, from a results perspective, is to determine how many virtual users a target system can support. While this virtual user information is useful, there are some fundamental issues regarding traffic pattern quality and target system bottleneck identification that are deemphasized or ignored.

These issues, if addressed, add credibility to the results produced and help product developers focus their performance improvement efforts. The virtual user script approach can work well as long as the traffic produced possesses a real world pattern and is quantifiable in target system terms; e.g., transactions per second. The emphasis should be on traffic pattern quality, traffic mix consistency, and traffic volume capacity not virtual user script count.

8. Summary

This document discusses the mechanics of traffic capacity testing transaction based applications running in a Unix/Linux environment. It explores some suggestions for quantifying the quality of the traffic being produced and more clearly articulating the traffic capacity of the target system. Some observations are made from these traffic generation and capacity specification remarks resulting in a set of recommendations and a list of conclusions.

9. References

[ALLE78] A.O. Allen, "Probability, Statistics, And Queueing Theory", Academic Press, Inc., Orlando, Florida, (1978).

[BRAD04], J.F.Brady, "Traffic Generation Concepts – Random Arrivals", www.perfdynamics.com, Classes, Supplements. (2004).

[COOP84] R. B. Cooper, "Introduction to Queueing Theory", Elsevier Science Publishing Co., Inc., New York, N.Y., (1984).

[GIFF78] W.C. Giffin, "Queueing: Basic Theory and Applications", Grid, Inc, Columbus, Ohio, (1978).

[KLEI75] L. Kleinrock, "Queueing Systems Volume 1 and 2", John Wiley & Sons, New York, N.Y., (1975).

[WHIT75] J.A. White, J.W. Schmidt, G.K. Bennett, "Analysis of Queueing Systems", Academic Press Inc., New York, N.Y., (1975).

Appendix A

Introduction

This appendix discusses the formulas used to produce negative exponential inter-arrival time traffic by first defining these formulas, second providing a graphical illustration of them, and third deriving them from first principles

Traffic Generation Formula

Either Equation A.1 or Equation A.2, below, is used to construct times between arrivals that adhere to the random arrivals requirement. Applying Equation A.2, the inter-arrival time, t_0, is obtained by multiplying minus the mean inter-arrival time, $-\mu$, by the natural logarithm of a random number between zero and one, $ln(r_0)$. Equation A.2 is generally used over Equation A.1 to obtain values of t_0 because $t_0 = -\mu\, ln(1 - r_0)$ and $t_0 = -\mu\, ln(r_0)$ yield symmetrical results and Equation A.2 requires one less arithmetic operation than does Equation A.1.

$$t_0 = -\mu\, ln(1 - r_0) \qquad \text{(A.1)}$$

$$t_0 = -\mu\, ln(r_0) \qquad \text{(A.2)}$$

Where:

$t_0 = time\ until\ the\ next\ arrival$

$ln = natural\ log\ (ln\, e = 1)$

$\mu = mean\ inter\text{–}arrival\ time$

$r_0 = random\ number\ 0 \le r_0 \le 1$

Traffic Generation Illustration

The graph and table in Figure A.1 provide motivation for using Equation A.1 or Equation A.2 to obtain independent inter-arrival times through an example with the mean inter-arrival time, $\mu = 5$.

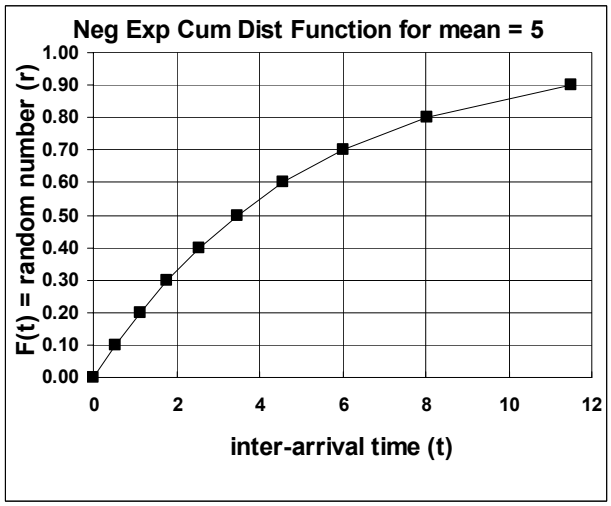
The technique used to produce the negative exponentially distributed inter-arrival times is the Cumulative Distribution Function (CDF) reverse transformation method implemented in many simulation software packages.

Figure A.1 illustrates this reverse transformation technique graphically. A uniformly distributed random number (0,1) is drawn and its location is found on the $F(t)$ axis. Then, a horizontal line is drawn from the $F(t)$ axis until it intersects the curve. The delay time until the next arrival is the t axis value at that intersection point. For example, if r = .50 is drawn at random the corresponding inter-arrival time is equal to 3.5.

Figure A.1: CDF Reverse Transformation

$$F(t) = Neg\ Exp\ Cum\ Dist\ Function(CDF)$$

$$r = F(t) = \int_0^t f(t)\, dt = \int_0^{t_0} \frac{1}{\mu} e^{-\frac{t}{\mu}}\, dt$$



t	F(t)
t = -u*ln(1-r)	r
0.0	0.0000
0.5	0.1000
1.1	0.2000
1.8	0.3000
2.6	0.4000
3.5	0.5000
4.6	0.6000
6.0	0.7000
8.0	0.8000
11.5	0.9000

As can be seen, the reverse transformation mechanism takes advantage of the fact that the range of the CDF for any probability distribution is zero through one. When random numbers are drawn from a uniform (0,1) probability distribution and mapped as described the resulting samples have the probability distribution characteristics of the CDF function implemented, which in this case, is the negative exponential.

Formula Derivation

What follows is a detailed derivation of Equation A.1 and Equation A.2. The first step is to integrate the negative exponential probability density function over the range $(0,t_0)$ as shown in Equation A.3 through Equation A.11. The second step, detailed in Equation A.12 through Equation A.18, is to set the results of this integral equal to a (0,1) random number, r_0, and solve for t_0, the desired time to the next arrival.

Step 1: Integrate the inter-arrival time probability density function over the range $(0,t_0)$:

$$f(t)=\frac{1}{\mu}e^{-\frac{t}{\mu}} \qquad \text{(A.3)}$$

$$F(t_0)=\int_0^{t_0} f(t)\,dt \qquad \text{(A.4)}$$

$$F(t_0)=\int_0^{t_0}\frac{1}{\mu}e^{-\frac{t}{\mu}}\,dt \qquad \text{(A.5)}$$

$$F(t_0)=\frac{1}{\mu}\int_0^{t_0}e^{-\frac{t}{\mu}}dt \qquad \text{(A.6)}$$

From Calculus:

$$\int e^{at}dt=\frac{e^{at}}{a} \qquad \text{(A.7)}$$

Evaluate the CDF at its limits $(0,t_0)$:

$$F(t_0)=\frac{1}{\mu}\left[-\mu e^{-\frac{t}{\mu}}\right]_0^{t_0} \qquad \text{(A.8)}$$

$$F(t_0)=\frac{1}{\mu}\left[\left(-\mu e^{-\frac{t_0}{\mu}}\right)-\left(-\mu e^{0}\right)\right] \qquad \text{(A.9)}$$

$$F(t_0)=\frac{1}{\mu}\left[\mu-\mu e^{-\frac{t_0}{\mu}}\right] \qquad \text{(A.10)}$$

$$F(t_0)=1-e^{-\frac{t_0}{\mu}} \qquad \text{(A.11)}$$

Step 2: set $r_0=F(t_0)$ and solve for t_0:

$$r_0=F(t_0)=1-e^{-\frac{t_0}{\mu}} \qquad \text{(A.12)}$$

$$r_0=1-e^{-\frac{t_0}{\mu}} \qquad \text{(A.13)}$$

$$e^{-\frac{t_0}{\mu}}=1-r_0 \qquad \text{(A.14)}$$

$$ln\left(e^{-\frac{t_0}{\mu}}\right)=ln(1-r_0) \qquad \text{(A.15)}$$

$$-\frac{t_0}{\mu}=ln(1-r_0) \qquad \text{(A.16)}$$

$$t_0=-\mu\,ln(1-r_0) \qquad \text{(A.17)}$$

Given the symmetry of $(1-r_0)\ and\ r_0$:

$$t_0=-\mu\,ln(r_0) \qquad \text{(A.18)}$$